Pilot-Data: An Abstraction for Distributed Data

Andre Luckow[1], Mark Santcroos[1,2], Ole Weidner[1], Ashley Zebrowski[1], Shantenu Jha[1*]

(1) *RADICAL, Rutgers University, Piscataway, NJ 08854, USA*
(2) *Bioinformatics Laboratory, AMC, University of Amsterdam, NL*
(*) *Contact Author:* shantenu.jha@rutgers.edu

ABSTRACT

The challenge of "Big Data" extends beyond the simple storage and management of large volumes of data. Science that involves and depends upon large amounts of data, also requires overcoming challenges in multiple other areas, including managing large-scale data distribution as well as co-placement and scheduling with computing resources. Although there exist multiple approaches to addressing each of these challenges, an integrative approach is missing; furthermore extending existing functionality or enabling cross-implementation capability for existing implementations remains difficult at best. To address the fundamental challenges of co-placement and scheduling of data and compute in heterogenous and distributed environments with interoperability and extensibility as first-order concerns, we define the concept of *Pilot-Data*, in analogy with, and symmetrical to, Pilot-Jobs. In this paper, we design and implement a Pilot-Data prototype, and deploy it on multiple production distributed cyberinfrastructure. We validate the concept of Pilot-Data by establishing that it provides a simple abstraction for managing data placement, whilst supporting interoperability across distributed environments and late-binding. Our experiments characterize the performance of an implementation of the Pilot-Data concept. They show a level of performance that is suitable for most large-scale scientific applications on (production) distributed cyberinfrastructure. We demonstrate how the concept of Pilot-Data also provide the basis upon which to build tools and support capabilities like affinity which in turn can be used for advanced data-compute co-placement and scheduling.

1. INTRODUCTION

Data has become a critical factor in many science disciplines [1]. As a consequence, the data generated by scientific applications, instruments and sensors is experiencing an exponential growth in volume, complexity and scale of distribution. The ability to analyze prodigious volumes of data requires flexible and novel ways to manage distributed data and computations. Furthermore, analytical insight is increasingly dependent on integrating different and distributed data sources, computational methods and resources [2].

Working with large volumes of data involves many challenges beyond its instantiation, storage and management. The specific challenge we address is that of co-placement and scheduling of data and compute in a distributed environment. The difficulty in being able to effectively and reliably manage co-placement and scheduling is in part due to the challenges inherent in distributed environments, compounded by an increasingly rich but incompatible and heterogeneous data-cyberinfrastructure incorporating diverse storage systems, data management semantics and multiple transfer protocols/approaches. Although these challenges have existed for a while, they are having progressively larger impacts on the performance and scalability of scientific applications; in particular, most currently available scientific applications still operate in legacy modes, e. g. they often require manual data management (e. g. the stage-in and out of files) and scheduling.

Some of the questions that arise include: (i) How to manage the placement, scheduling and distribution of data effectively so as to be available when needed by a computational task? (ii) What are the right abstractions for coupling compute and data that hold for a range of application types and infrastructures? Furthermore, how can both system-level and application-level information be incorporated to support dynamic and late-binding of compute and data to resources? (iii) Last but not least, how can the inherent heterogeneity be handled? How can one provide an interoperable, uniform access to these heterogeneous distributed data-cyberinfrastructure and resources?

These challenges are common to many scientific applications. In this paper we focus on two well known exemplars: climate modeling as performed by the Earth System Grid Federation (ESGF) and Next-Generation Sequencing analysis applications [3]. But before that, we mention that there exist multiple other challenges viz., data security, data access rights and policy, and data semantics and consistency. These are all important determinants of the ultimate usability and usage-modes but we will not consider them to be in scope of the work of this paper. Our decision is in part explained by the fact that our work is ultimately aimed towards the development of abstractions and middleware for production distributed cyberinfrastructure (DCI) which will be agnostic to specific security and data-sharing policies."

Application Exemplars

ESGF: The overall data generated and stored is 2-10 PB, of which the most frequently used data is 1-2 PB in size. The data is generated by a distributed set of climate centers, and

it is stored in a distributed set of federated archives. The data is used by a distributed set of users, who either run data analyses on a climate center with which they are associated, or they gather data from the ESGF to a local system for their analyses. Furthermore, data which is generated over time causes real-time changes — spatial and temporal, in the ESGF dataset; the scheduling of data analysis jobs needs to be responsive to these spatio-temporal data changes.
NGS analytics: In addition to being a big data problem (O(Terabytes)), NGS analytics is also a computationally demanding and distributed computing problem. The computational demands arise from the often complex and intensive analysis that has to be performed on data, which in turn arise from algorithms that are designed to account for repetitions, errors and incomplete information. The distributed aspects arise at multiple levels: for example, the simple act of having to move data from source (generation) to the destination where computing (analysis) will occur is a challenge due to the volumes involved. Trade-offs exists between the cost/challenges in distributing data versus I/O saturation or memory bottlenecks. When coupled with the compute intensive nature of the problem, it soon emerges that a fundamental challenge is not only whether to distribute, but where to distribute, what to distribute (should the computing move to the data, or the data move to the compute), and how to distribute (what tools and infrastructure to use).

The execution of both ESGF and NGS applications on distributed infrastructure can have dynamic aspects when optimized resource usage is considered. For example, workload decomposition and distribution must be determined dynamically in order to optimally use resources, e.g., compute resource selection can be based on data location, processing profile and/or network capacity.

Overview and Outline

As the ESGF and NGS analytic examples establish, the real bottleneck is the collective distributed compute-data management and scheduling problem. Pilot-Jobs have a demonstrable record of effective distributed resource utilization and supporting a broad range of application types [4], [5]. In this paper, we explore the generalization of the Pilot-Job concept, via Pilot-Data, to that of Pilot-Abstractions as a way to efficiently manage distributed data and its dynamic placement with respect to computation.We associate our work with Pilot-Jobs so as to take advantage of their established capability of providing distributed computing resources. By doing so, we also alleviate a significant gap that has come to the fore as typical data-volumes used/required by science applications has increased, viz., there is insufficient and incomplete support for data movement and placement for many Pilot-Job implementations [4].

Specifically, we introduce *Pilot-Data (PD)* as a novel abstraction for data-intensive applications that provides late-binding capabilities for data by separating the allocation of physical storage and application-level Data-Units. Although there are multiple pre-existing approaches that dynamically couple data to compute tasks, there are two distinct features of the Pilot-Data approach: first, Pilot-Data provide a mechanism that can work in conjunction with Pilot-Jobs but is not dependent upon them. Second, Pilot-Data provides a general approach to data-compute placement, in that it is not constrained to a specific scheduling algorithm or infrastructure; the only potential limitation in arises as a consequence by the Pilot-Job that it may be paired with.

The role of a conceptual abstraction is to be able to reason about a capability without having to worry about implementation details of that capability. Specifically, the suggestion that Pilot-Data is a conceptual abstraction for distributed data is predicated upon the fact that like any valid abstraction, it must provide applications with a unifying programming model and usage modes. As an example of how this is achieved, Pilot-Data provides a simple and useful notion of distributed logical location that from an application's perspective is invariant over the lifetime; thus it supports both a decoupling in time (i. e., allowing late-binding) and space between actual physical infrastructure and the application usage of that infrastructure. Pilot-Data must thus retain the flexibility to be used with different CI whilst not constrained to different specific modes of execution or usage.

A critically important point to note is that our focus is on addressing the challenges outlined in the context of *production distributed CI* and not research prototypes, or "feature rich" but closed or specific data-cyberinfrastructure and back-end systems. We believe there are three primary macroscopic architectures for scalable production distributed cyberinfrastructure: (i) the first when the data is essentially localized, e.g., "poured" into a cloud, or given the volumes of data, the scale over which localization occurs is relatively small. Interestingly, given the ratio of the high computational and data-storage capacity to the number of sites in XSEDE, it too can be classified in this category; (ii) where the data is decomposed and distributed (with multi-tier redundancy and caching) to an appropriate number of computing/analytical engines as available, e. g., as employed by the European Grid Initiative (EGI) [6]/Open Science Grid (OSG) [7] for particle physics and the discovery of the Higgs, and (iii) a hybrid of the above two paradigms, wherein data is decomposed and committed to several infrastructures, which in turn could be a combination of either of the first two paradigms.

We will show how Pilot-Abstractions can be used for all three architectural paradigms. However, it is not enough to suggest that an abstraction is compatible with a macroscopic architectural paradigms, for there can be multiple implementations of a macroscopic architecture, e. g., OSG and EGI are very different implementations of essentially the same architecture, with significant differences in the middleware, tools and services available.

Combining the previous points form the basis for our claims that Pilot-Data, when taken in conjunction with Pilot-Jobs, provides a unified Pilot-Abstractions and presents a general and flexible solution to the collective distributed compute-data management and scheduling problem. Furthermore, our model and implementation of Pilot-Abstractions is fundamentally agnostic to the underlying CI; our implementation interfaces seamlessly with SAGA [8], which is a well established route to interoperability.

In §2 we discuss related work with a view to providing the reader with a better appreciation for the scope of our work with respect to other distributed data scenarios. Before discussing the Pilot-abstractions, we present a simple but general model in §3, to understand the primary components and trade-offs to determine compute-data placement; this model is independent of any specific infrastructure or approach. We begin §4 by discussing Pilot-Jobs and the P* Model – a minimal and complete model for Pilot-Jobs;

we discuss how the P* Model supports a logical extension to Pilot-Data as an abstraction to support distributed data scenarios. §5 discusses data-compute placement in the context of Pilot-Jobs and Pilot-Data and introduces a Pilot-API as means of expressing the coupling of these entities. In order to establish and evaluate Pilot-Data as an abstraction for distributed data, we design and conduct a series of experiments in §6. We conclude with a discussion of the main lessons learned as well as relevant and future issues.

2. RELATED WORK

The landscape of solutions that have been devised over the years to address the challenges and requirements of distributed data is very vast. Any discussion of relevant related work has to be focussed by design and limited by necessity. Thus in this section we provide a brief discussion of relevant efforts that either support compute-data co-placement and/or support data management in production DCI, such as XSEDE, OSG and EGI.

2.1 Distributed Data Management

A myriad of storage solutions exist, ranging from local and parallel filesystems, e. g. Lustre [9], to distributed data stores, such as Amazon S3 [10]. Commonly, storage systems can be accessed via different protocols, e. g. the virtual filesystem layer in Linux or a transfer protocol, such as GridFTP [11]. Often, these storage systems provide unknown quality of services, i. e. an application is only aware of high-level characteristics of a system and usually does not know a-priori what throughput and latencies it can expect.

Several distributed data management systems have been built on top of these low-level storage systems to facilitate the management of geographically dispersed storage resources. Systems like SRM [12] and iRODS [13] combine storage services with services for metadata, replica, transfer management and scheduling. Also, different services covering singular aspects, such as replica management (e. g. the Replica Location Service (RLS) [14] or the LCG File Catalogue (LFC) [15]), exist. Globus Online [16] is a hosted transfer service that is based on GridFTP. In the following we focus on some representative examples.

Storage Resource Manager (SRM) is a type of storage service that provides dynamic file management capabilities for shared storage resources via a standardized interface. SRM is primarily designed as an access layer with a logical namespace on top of different site-specific storage services. SRM aims to hide the complexity of different low-level storage services, but does not allow applications to control and reason about data locality; this is left to other components, e. g. a combination of information system and replica manager (BDII [17] & LFC in the case of EGI).

iRODS is a comprehensive distributed data management solution designed to operate across geographically distributed, federated storage resources. Central to iRODS are the so called micro-services, i. e. the user defined control logic. Micro-services are automatically triggered and handle pre-defined tasks, e. g. the replication of a data set to a set of resources.

The Global Federated Filesystem (GFFS) [18] is part of the Genesis II middleware and provides a global namespace on top of a heterogeneous set of storage resources. The system handles file movement and replication transparently.

2.2 Pilot-like Approaches for Distributed Data

Pilot-Jobs have been successful abstractions in distributed computing as evidenced by a plethora of PJ frameworks: Condor-G/Glide-in [19], DIANE [20], PanDA [21], ToPoS [22], Nimrod/G [23], Falkon [24] and MyCluster [25], to name a few. However, only a few of them provide integrated compute/data capabilities. For example, DIRAC [26] is a Pilot-based workload management system for compute and data, which is used in the LHC Computing Grid. The integrated data management system maintains replica locations and manages data transfers. For this purpose, the system interfaces to SRM storage resources. Another example is Swift [27], which provides a data management component called CDM. DIANE provides in-band data transfer functionality over its CORBA channel.

2.3 Data-Compute Co-Scheduling

There are several projects that aim towards integrated data and compute scheduling. For example, the Stork [28] data-aware batch scheduler provides advanced data and compute placement for Condor and DAGMan. Stork supports multiple transfer protocols like, SRM, (Grid)FTP, HTTP and SRB. Romosan et al. [29] present another data-compute co-scheduling approach on top of Condor and SRM. Both approaches build on top of existing job scheduling and data-transfer and storage solutions.

Another well-known example is the MapReduce framework Hadoop [30]. Hadoop provides a highly integrated environment for data-intensive applications. The new Hadoop resource scheduler, Yarn [31], tightly integrates the Hadoop filesystem with the compute resources of the cluster. Compute tasks are whenever possible placed at the same node as the data. Similar integrated capabilities are currently missing from the scientific cyberinfrastructures. While Hadoop gained a lot of traction in local cluster environments, it lacks the ability to efficiently handle distributed data.

Various abstractions for optimizing access and management of distributed data have been proposed: Filecule [32] is an abstraction that groups a set of files that are often used together, allowing an efficient management of data using bulk operations. This includes the scheduling of data transfers and/or replications. Similar file grouping mechanisms have been proposed by Amer et al.[33], Ganger et al. [34] and BitDew [35]. Another example is DataCutter [36] a framework that enables exploration and querying of large datasets while minimizing the necessary data movements.

Finally, different research on when to (potentially dynamically) distribute and replicate data has been conducted: for example, Foster [37] and Bell [38] investigate different data replication management system and dynamic replication algorithms in the context of scientific data grids. A limitation of the previous approaches is that the systems and algorithms are usually constrained to system-level replication, making it difficult for the user to control replication on application-level and employ dynamic replication strategies.

A limitation of current data-cyberinfrastructures is the lack of integration between data and compute capabilities. Many systems solely focus on data or compute aspects leaving it to the application to integrate compute and data. Typically, this leads to the necessity to manually move data in and out of the systems. Localities between data and compute infrastructure are often unknown and difficult to de-

duce, thereby making it difficult for the application to optimize data and compute placements. System-level schedulers are usually not aware of the application characteristics (e. g. specific compute/data dependencies), which usually leads to non-optimal placement decisions. In many cases there is a need to manually allocate and manage resources on a very low level in order to achieve acceptable performance.

3. A SIMPLE MODEL FOR COMPUTE-DATA PLACEMENT

A question that arises in the design of systems and that distributed data-intensive applications have to address, is whether to assign and move computational tasks to where data resides, or to move data to where computational tasks are executed. Another fundamental question is when to commit to a given approach. Additionally, if replication is an option, applications and systems have to determine what the degree of replication of data should be, and possibly where to replicate. Here we present a simple model to help reason along the above lines. We posit that the fundamental parameters in an infrastructure agnostic model are:

- T_Q defined as the queue waiting time at a given resource.
- T_C defined as the compute time.
- T_X defined as the time to transfer a number of bytes, say B bytes, over the network to a given a resource from a defined source; this measures the time-in-flight for data.
- T_S is the staging time, which is defined as the data time-in-flight plus the time to get data into a system (i.e. register). Staging time could be for either the upload time or download time, thus when referring to upload, $T_S = T_U$. As T_S is is defined as the sum of T_X and $T_{register}$, where the latter time is defined as the time to register data into the end points (e.g., a thousand files into a catalogue). For most scenarios in this paper, $T_{register}$ is negligible compared to T_X to a first order, thus $T_X = T_S$.
- T_R defined as the time to replicate data, where R is the number of sites that data is replicated over.
- T_D is the time at which data will be accessible across all distributed resources. When replication is involved, it is defined as the sum of the $T_R(R)$ and T_S.

The above expression provides a basis to reason whether to distribute/replicate data before determining where to compute, or to distribute/replicate data after having determined which compute resources to use. To a first approximation, which of the two approaches should be employed is given by the relative values of T_D and the typical value of T_Q, which in turn has a dependence on the data volume under consideration. The desired mode is also strongly dependent upon data volumes in considerations, as well as the capability of the tools and middleware in use, however, it is important to reiterate that our model is agnostic of specific implementation details. As an example, although our focus is on Pilot-Abstractions the model per se does not presume any dependence on Pilots.

We begin by considering the simple scenario, where data replication is not an option (Case A). Thus, the only non-zero contribution to T_D is provided by T_X. In this case the decision about which entity (D or C) to place/schedule first and which to move – compute to data (C2D) or data to compute (D2C) – is determined via a simple trade-off between T_Q and T_X. Assuming there is only one site to choose, the relative values of the previous two terms is determined for all possible sites; the site with the greatest similarity between the mean value of queue waiting time and T_X is chosen. If T_X is larger than the mean, then the compute is assigned to a site first, and subsequently data is placed. Although significantly more complex, the above can be generalized to schedule/place multiple tasks over multiple sites. Having established that, we move our attention to the case when replication is an option, i. e., Case B.

For Case B, an *a priori* distribution (and or replication) of data can be assumed to exist; in response to this distribution, compute resources are chosen. Consequently, from the list of resources co-located with a data replica, the resource with the lowest queue waiting time presents an optimal choice. However, it is important to appreciate that there is an overhead in ensuring that data is replicated in a distributed fashion (which we refer to as the upload phase). Again, moving compute to data (respectively close to the data) vs. data to compute is a degree of freedom.

In practice hybrid modes can be employed. As an example, distributed data replication can initially be set to be partial, viz., only over a sub-set of possible distributed sites. In other words, replication might commence over a sub-set of suitably chosen nodes, followed by a sequential increase in the replication (factor) if compute resources close to the replica do not have sufficient compute capacity.

4. PILOT-ABSTRACTIONS: A UNIFIED ABSTRACTION FOR COMPUTE AND DATA

The seamless uptake of distributed infrastructures by scientific applications has been limited by the lack of pervasive and simple-to-use abstractions at multiple levels – at the development, deployment and execution stages. A survey of actual usage suggested that Pilot-Jobs were arguably one of the most widely-used distributed computing abstractions [4] – as measured by the number and types of applications that use them, as well as the number of production distributed CI that support them. Although Pilot-Jobs have been used by many high-throughput applications, there does not exist a well defined, unifying conceptual framework for Pilot-Jobs which can be used to define, compare and contrast different implementations. Our survey led us to understand that different Pilot-Jobs had different semantics and capabilities, and made vast if not inconsistent assumptions of applications/users. This presented a barrier to usability and extensibility of Pilot-Jobs.

To address this barrier, we have recently developed [4] the Pstar (P*) model for Pilot-Jobs, which provides a minimalistic but the first complete model of Pilot-Jobs, with the functional objective being to provide a single conceptual framework that would be used to understand and reason about different Pilot-Job implementations. Furthermore, we have implemented BigJob (BJ) [5] – a SAGA-based Pilot-Job, as a reference implementation of the P* model. By linking the theoretical underpinnings and implementation to its usage, we have established Pilot-Jobs as a richer and more powerful runtime environment than was hitherto considered possible or realized. For example, until now Pilot-Jobs have essentially been "passive" elements – not programmable, nor as a means of expressing specific application requirements. The P* model promotes Pilot-Jobs to an "active" element, capable of supporting application specific scheduling and placement requirements, as well as coordination and coupling requirements. This in turn has led to a break from traditionally constrained and confined usage modes for Pilot-Jobs,

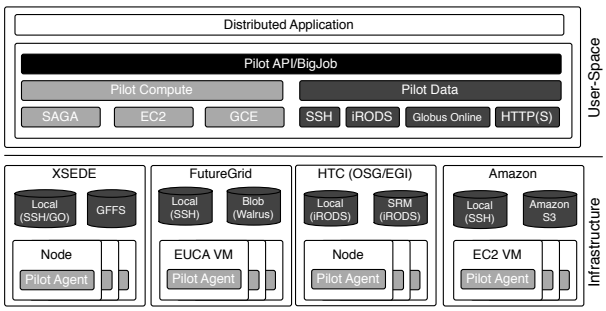

Figure 1: BigJob Pilot-Abstractions and Supported Resource Types: BigJob provide a unified abstraction to a heterogeneous set of distributed compute and data resources. Resources are either accessed via SAGA [8, 41] or via a custom adaptor.

as evidenced by a step-change that has arisen in the nature and scale of problems using Pilot-Jobs [39].

The P* model defines Pilot-Computes (PC) as the fundamental entity that is submitted to the resource. The application workload is described using so called Compute-Units (CUs) and submitted via the Pilot-Compute. The notion of Pilot-Data (PD) was conceived using the power of symmetry, i. e., the notion of Pilot-Data was as fundamental to dynamic data placement and scheduling as Pilot-Jobs was to computational tasks. As a measure of validity, the P* model was amenable and easily extensible to Pilot-Data. The consistent and symmetrical treatment of data and compute in the model led to the generalization of the model as the *P* Model of Pilot Abstractions*. PD provides late-binding capabilities for data by separating the allocation of physical storage and application-level Data-Units. Similar to a Pilot-Compute, a Pilot-Data provides the ability to create Pilots and to insert respectively retrieve Data-Units. The Pilot-API [40] provides an abstract interface to both Pilot-Compute and Pilot-Data implementations that adhere to the P* model.

4.1 BigJob: A Pilot-Compute and Data Implementation

A main limitation of Pilot-Job frameworks is the inability to manage distributed data. From a practical point-of-view, data management and movement for most Pilot-Jobs is at best obscure if not outright ad hoc. Most Pilot-Jobs rely on application-level data management, i. e. data needs to be pre-staged or each CU is responsible for pulling in the data. In the following, we propose Pilot-Data as a consistent interface for data management in conjunction with Pilot-Jobs. Similarly to Pilot-Jobs, Pilot-Data is an application-level construct that allows the logical decoupling of physical storage/data locations and the production and consumption of data. Among many things, Pilot-Data facilitates late binding of data and physical resources. The Pilot-Data-Abstraction aims to support the distributed coupling of different application components, i. e. the Compute-Units and Data-Units, e. g. the parts of a distributed workflow or the data flow between a compute and analysis job.

Pilot-Data is an extension of BigJob (BJ) [5, 42], which provides a unified runtime environment for Pilot-Computes and Pilot-Data on heterogeneous infrastructures (see Figure 1), along with a higher-level, unifying interface to heterogeneous and/or distributed data and compute resources. As Pilot-Jobs eliminate the need to interact directly with different kinds of resources, e. g. batch-style HPC/HTC or

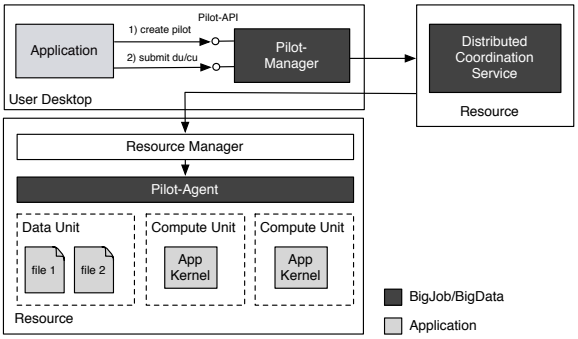

Figure 2: BigJob High-Level Architecture: The Pilot-Manager is the central coordinator of the framework, which orchestrates a set of Pilots. Each Pilot is represented by a decentral component referred to as the Pilot-Agent, which manages the set of resources assigned to it.

cloud resources, Pilot-Data removes the necessity to manually interoperate with different data sources and stores, e. g. file storage, repositories, databases etc, by providing a unified data management and access layer, which can be used to allocate and access a heterogeneous set of resources.

Figure 2 shows the high-level architecture of BigJob. The Pilot-Manager is the central entity, which manages the actual Pilots via the Pilot-Agent. BigJob supports different resource types via an adaptor mechanism (see adaptor pattern [43]). The adaptor encapsulates the different infrastructure-specific semantics of the backend system, e. g. in the case of Pilot-Data different storage types (e. g. file vs. object storage), access and transfer protocols. Figure 1 illustrates the available adaptors. For Pilot-Computes, BJ supports various types of HPC/HTC resources via SAGA-Python [41] (e. g. Globus, Torque or Condor resources). Further, adaptors for cloud resources (Amazon EC2 and Google Compute Engine) exist. Similarly, Pilot-Data can marshal different types of storage resources and access protocols, e. g. a local filesystem accessed via SSH or a transport service, such as Globus Online [16], which again utilizes GridFTP [11] as transport protocol. Also, Pilot-Data can support more advanced distributed data management systems, such as iRODS. The respective resource adaptor is selected based on the resource URL to the backend system defined by the application in the Pilot-Description.

Figure 3 shows the typical interactions between the components of the BigJob/Pilot-Data framework after the submission of the application workload (i. e. the CUs and DUs). The core of the framework is the Pilot-Manager. The Pilot-Manager is able to manage multiple Pilot-Agents. The application workload is submitted to the Pilot-Manager via the Compute-Data Service interface of the Pilot-API (see section 4.2). After submission, DUs and CUs are put into a in-memory queue, which is continuously processed by the scheduler component. This asynchronous interface ensures that the application can continue without needing to wait for BigJob to finish the placement of a CU or DU.

Another core component is the distributed coordination service that is used for communication between the Pilot-Manager and Pilot-Agent. Currently, BigJob relies on Redis for this purpose. Other kinds of coordination services are supported via a plugin scheme. The main task of Redis is to facilitate the communication and coordination between Pilot-Manager and Agents: (i) The Pilot-Agent collects var-

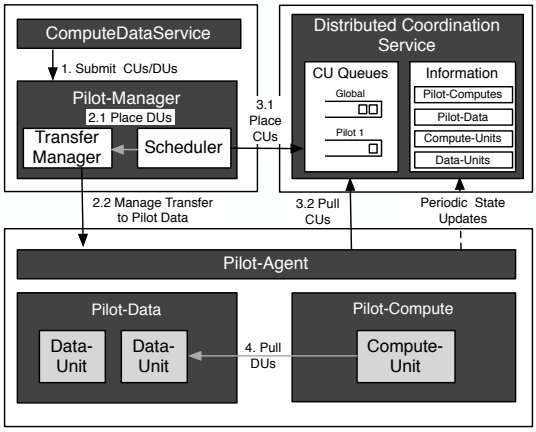

Figure 3: BigJob Application Workload Management: The figure illustrates the typical steps involved for placing and managing the application workload, i. e. the DUs and CUs.

ious information about the local resource, which is is pushed to the Redis server and used by the Pilot-Manager to conduct e. g. placement decisions; (ii) CU are stored in several queues. Each Pilot-Agent generally pulls from two queues: its agent-specific queue and a global queue. Since the Redis server is globally available, it also serves as central repository that enables the seamless usage of BigJob from distributed locations. That means that application can connect back to its entities (i. e. Pilots, DUs, and CUs) via a unique URL and query it for state information and update it.

4.2 Pilot-API: Managing Distributed, Data-Intensive Resources and Computations

The Pilot-API [40] provides a well-defined control and programming interface for pilots, which is built upon the consistent and well-defined semantics associated with pilots [4]. It is an interoperable and extensible API which exposes the core functionalities of a Pilot framework via a unified interface providing a common API that can be used across multiple distinct production cyberinfrastructures . It offers a unified API for managing both compute and data pilots as well as application workloads. BigJob provides a full implementation of the Pilot-API and enable the management of resources, CUs & DUs as well as the relationships between them. Specifically, the Pilot-API promotes affinities as a first class characteristic for describing such relationships between compute and data elements defined by the P* model and to support dynamic decision making.

The API separates the concerns (i) management of pilots and resources and (ii) application workload management (see Figure 4). The Pilot-Compute Service is responsible for controlling the lifecycle of Pilot-Computes; the Pilot-Data Service manages Pilot-Data. Using Pilot-Abstractions the different types of distributed compute resources, storage infrastructures and transport protocols etc. can be marshaled providing a seamless, unifying environment to the application for managing resources. Applications can simply acquire resources in form of Pilots and then assign their workload to these resources.

The application or tool that uses the Pilot-API utilizes Data-Units (DUs) and Compute-Units (CUs) as abstraction for expressing the application's workload. A CU represents a

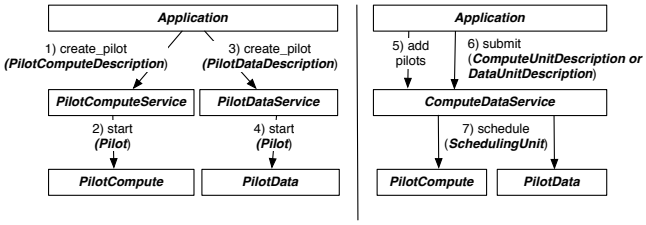

Figure 4: Control Flow Pilot-API: The API exposes two primary functionalities: The PilotComputeService and PilotDataService are used for the management of Pilot-Computes and Pilot-Data. The application workload is submitted via the Compute-Data Service.

primary self-containing piece of work, while a DU represents a self-contained, related set of data. A DU is a container for a logical group of "affine" data, e. g. data that is often accessed together by various Compute-Units. A Compute-Unit is a computational task that potentially operates on a set of input data, specified in form of one or more dependent Data-Units.

The Compute-Data Service is the central abstraction for submitting and managing this application workload, i. e. the Compute-Units and Data-Units. Both CUs and DUs are submitted to the runtime system via the Compute-Data Service interface. The runtime system is responsible for placing CUs and DUs on a Pilot (placed on a respective resources). For this purpose, the runtime system relies e. g. on the localities of the DUs, to facilitate scheduling and other types of decision making (see section 5).

4.3 Experiences on Different Distributed Infrastructures

The objectives of the following experiment is to demonstrate the ability of Pilot-Data to marshal different storage backend infrastructures. Using Pilot-Data we will attempt to characterize the behavior of PD on different cyberinfrastructures (e. g. XSEDE and OSG) before we investigate advanced compute/data placements strategies in section 6.

For our experiments, we utilize GW68 – a gateway node located at Indiana University and part of the XSEDE infrastructure – as our submission machine to both XSEDE and OSG resources. Figure 5 illustrates T_U for different backends, i. e. the time necessary to populate a Pilot-Data store on different infrastructures, i. e. (i) a directory of the work filesystem on Lonestar, (ii) a S3 bucket and (iii) iRODS on OSG. For scenario (i), we utilize different data movement systems to access the data: SSH and Globus Online/-GridFTP. For all scenarios, we only consider the T_U part of T_D (which is equivalent to T_S), i. e. for the iRods scenario T_R is not depicted.

Figure 5 illustrates the results of this experiment. The performance of Pilot-Data primarily depends on the infrastructure used. In general, the overhead of Pilot-Data itself is very low. T_U is dominated by T_X, i. e. the time necessary to transfer files to the Pilot-Data location. Experiments with smaller data sizes have shown that $T_{register}$ is negligible. Thus, the runtime is directly influenced by the available bandwidth and the characteristic of the respective transfer protocol. While for smaller data sizes SSH performs best, Globus Online particularly performs well for larger sizes; in particular by relying on GridFTP as transfer protocol. T_U for iRods behaves comparable to T_U for SSH. T_U for S3 in-

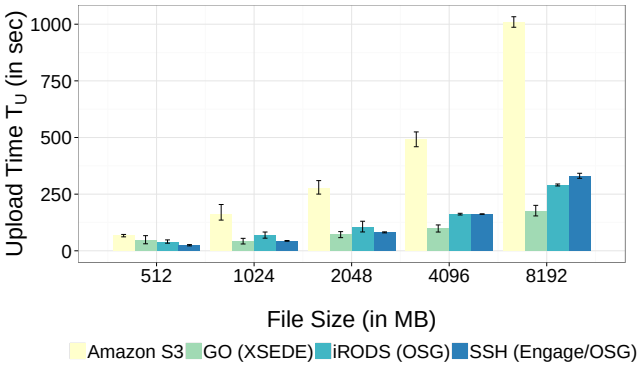

Figure 5: Pilot-Data on Different Infrastructures: Time to initialize a Pilot-Data with a dataset of given size. For iRods, we measure only the upload time and not the time required to replicate the data. SSH/iRods perform well for small data sizes; Globus Online for larger data volumes. S3 performance is limited by Internet connectivity.

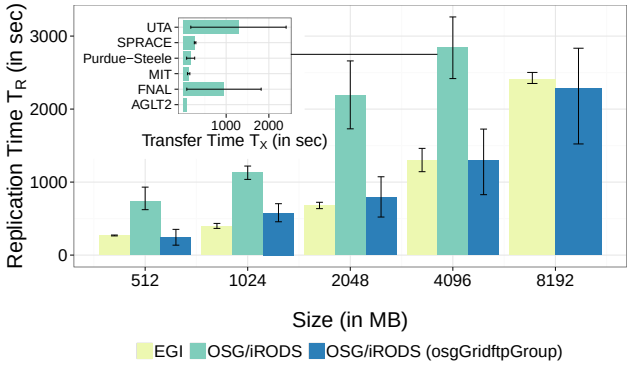

Figure 6: Replication on EGI and OSG: T_R on different infrastructures and resource sets: on EGI to 11 SRM resources of the Dutch e-Science Grid, in the OSG scenario to 6 iRODS resources and in the OSG (osgGridFTPGroup) scenario to 9 iRODS resources that are members of this group. The inset shows the distribution of T_R with respect to the different hosts for the 4 GB & OSG/iRODS scenario.

creases linearly – an indicator that it is bound to the available Internet bandwidth.

If system-level support for replication is provided, e. g. by a distributed data management middleware such as iRods, Pilot-Data can utilize this capability as a dynamic caching mechanism (to be contrasted with the usage of iRODS for data storage and management). While we previously only considered T_U, we continue our investigation with an evaluation of T_R, i. e. the replication overhead. In the following experiment, we investigate T_R for different infrastructures and configurations: (i) iRODS/OSG with group-based replication, (ii) iRODS/OSG with sequential replication in which one replica is created after the other, and (iii) EGI with sequential replication using SRM/LFC. For this purpose, the data is replicated to resource sets of different types and sizes: on EGI to 11 SRM resources of the Dutch e-Science Grid, in the OSG scenario to 6 iRODS resources and in the OSG (osgGridFTPGroup) scenario to 9 iRODS resources that are members of this group.

Figure 6 illustrates the results. On OSG the T_R for the group-based replication is significantly better than for the sequential replication. In both cases, the frequency of failures was very high. While the `osgGridFtpGroup` group consisted of 9 nodes, the average number of resources that actually received a replica was ∼7.5. In general, the overall performance is determined by the available bandwidth between the central iRODS server (located at Fermilab near Chicago) and the individual sites. For 4 GB case in scenario (ii), the individual T_X are depicted in the inset of Figure 6. While the performance of the EGI scenario is very good, it must be noted that the resources are not as geographically dispersed as in the OSG case, where resources are distributed across the east, south and center of the US. Nevertheless, the sequential replication is well suited for creating a small number of replicas. The different sites have very different performance characteristics. Thus, the ability for applications to optimize data/compute placement with respect to their computational and data requirement is critical. If the compute resources of the three "closest" sites are sufficient and available, it is e. g. not necessary to replicate all data across OSG.

5. DATA PLACEMENT AND COMPUTE-DATA CO-SCHEDULING

The aim of Pilot-Data is to enable the efficient management of data in conjunction with compute. Different investigations (e. g. [44, 37]) have shown that when considering data/compute entities equally while making placement decisions leads to performance gains. An important consequence of data and computation as equal first-class entities, is that either data can be provisioned where computation (D2C) is scheduled to take place (as is done traditionally), or compute can be provisioned where data resides (C2D). This equal assignment of Compute-Units leads to a richer set of possible correlations between the involved DUs and CUs; correlations can be either spatial and/or temporal. These correlations arise either as a consequence of constraints of localization (e. g., data is fixed, compute must move, or vice-versa), or as temporal ordering imposed on the different data and computational units.

We propose affinities as an abstraction for managing co-locations of Pilot-Data, Pilot-Computes and DUs/CUs in section 5.1. We continue with a description on how PJ frameworks, such as BigJob, can use affinities for making data/compute placement decisions in section 5.2.

5.1 Affinities: Managing Relationships between Data and Compute

Managing data locality in a distributed environment is a great challenge due to the heterogeneous landscape of data/-compute infrastructures coupled with the large amount of dynamism associated with distributed environments. Traditionally, Pilot-Abstractions have been used to provide a uniform interface to these heterogeneous infrastructures. We propose affinities as an extension to Pilot-Abstractions. Affinities are an essential tool for modeling different resource topologies, application characteristics and allow for effective reasoning about resource topologies and data/compute dependencies. We will show in section 6 that the usage of affinities in conjunction with Pilot-Abstractions for distributed data will yield to better performance and scalability compared to simplistic if not ad-hoc ways of data-compute cou-

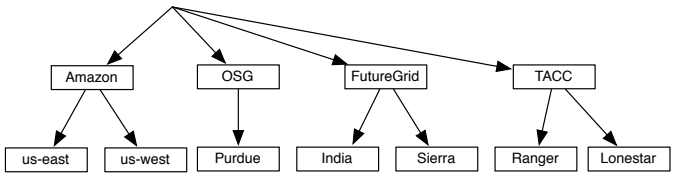

Figure 7: Affinities between Distributed Resources: Pilot-Data assigns each resource an affinity based on a simple hierarchical model. The smaller the distance between two resources, the larger the affinity.

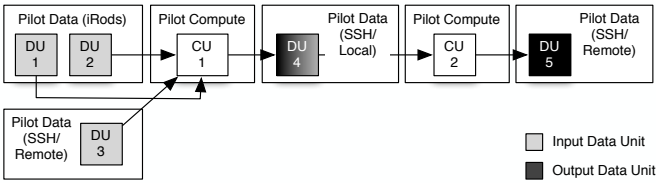

Figure 8: DU and CU Interactions and Data Flow: Each CU can specify a set of input DU. The framework will ensure that the DU is transferred to the CU. Having terminated the job, the specified output is moved to the output DU.

pling while still maintaining a high level of simplicity.

The concept of affinity is used to describe the relationship between the different entities of a PJ-Framework (defined in the P* model [4]), e. g. the relationship between CUs, DUs, Pilots and resources. This model enables the framework to reason about different trade-offs, e. g. to make decisions on whether on to move data versus move the compute (see section 5.2) based on resource and bandwidth availabilities.

The model supports different kinds of affinities: (i) Pilot/resource affinities describe the relationship between multiple pilots and the resources they are running on; (ii) compute/data affinities describe the relationships between CUs and DUs, which can exist in various forms, e. g. D-D, C-D, or C-C.

Resource/Pilots Affinities: Resource affinities describe the relationship between a set of compute and/or storage resources. We use a simple model for describing resource affinities: Data centers and machines are organized in a logical topology tree (similar to the tree spawned by an DNS topology). The further the distance between two resources, the smaller their affinity. Figure 7 shows e. g. how a distributed system consisting of different types of cloud and grid resources can be modeled. Using such an resource topology, the runtime system can deduce the connectivity between two resources, to estimate e. g. the costs induced by a potential data transfer. While this model is currently very coarse grained, it can be enhanced by assigning weights to each edge to reflect dynamical changes in factors that contribute to connectivity .

The affinity of Pilot is determined based on the resource it is located on, i. e. the proximity of two Pilots is deduced from the distance of their resource in the resource topology tree. The mapping between resource and Pilot is done by assigning each Pilot a logical location using a user-defined affinity label in the Pilot-Description. This logical location assignment is utilized by the scheduler to create the resource topology tree.

Compute/Data Affinities: As described, applications utilize DU as a primary abstraction for grouping data. CUs can have input and output dependencies to a set of DUs, i. e. the data of these DUs is required for the computational phase of the CU. The output data is automatically written to one or more output DUs. The framework utilizes these affinities to place DUs and CUs into a suitable Pilot-Compute or Pilot-Data. Further, CUs and DUs can constrain their execution resource to a particular affinity (e. g. to a certain location or sub-tree in the logical resource topology). The runtime system then ensures that the data and compute affinity requirements of the CU/DU are met.

Figure 8 shows typical data/compute dependencies and a typical data flow between multiple phases of compute. Applications can declaratively specify CUs and DUs and effectively manage the data flow between them using the Pilot-API. A CU can have as described input and output dependencies to a set of DUs. For this purpose, the API declares two fields in the Compute-Unit Description: `input_data` and `output_data` that can be populated with a reference to a DU. The runtime system ensures that these dependencies are met when the CU is executed, i. e. either the DUs are moved to a Pilot that is close to the CU or the CU is executed in a Pilot close to the DU's Pilot. In the best case, the Pilot-Data of the depended DUs is co-located on the same resource as the CU, i. e. the data can be directly accessed via a logical filesystem link. Otherwise, the data is moved via a remote transfer. Further, a CU can constrain its execution location to a resource with a certain affinity. If specified, the scheduler will only place the CU in a Pilot with the right affinity.

The input data is made available in the working directory of the CU. As described, depending on the locality of the DU/CUs, different costs can be associated with this operation. The runtime system relies on an affinity-aware scheduler that ensures that data movements are minimized and that if possible "affine" CUs and DUs are co-located (see section 5). Typically, scientific applications also involve multiple steps of data generation and processing. As depicted in Figure 8, the Pilot-API can be used to efficiently manage data flow between a set of dependent CUs. The input data can be pulled in from a remote resource or a data management system, such as iRODS. Intermediate data can be stored locally for further processing in a second stage (where a filtering and/or aggregation can take place) before the data is moved to a remote resource.

5.2 Pilot-based Scheduling

The Pilot-Abstraction provides the basis for application-level scheduling. Applications have two options: (i) rely on the internal scheduler of the PJ framework (i. e. BigJob) or (ii) make placement decisions based on the Pilots it has acquired, i. e. it can manually place CUs and DUs in specific Pilots. The latter gives an application a high degree of freedom, allowing it to pursue strategies optimal for itself, e. g. by choosing the right numbers of replicas for a certain amount on compute that needs to be carried out on the data.

The scheduler of BigJob is a plug-able component of the runtime system and can be replaced if desired. The default implementation referred to as the affinity-aware scheduler relies on the resource topology information specified via the Pilot-API. This scheduler is hidden behind the Compute-Data Service interface. It uses the specified affinities to reason about the relationships between DUs, CUs, Pilots and resources to optimize data localities and movements.

The affinity-aware scheduler currently implements a sim-

ple strategy based on earlier research [44] that suggests that considering both data and compute during placement decisions leads to a better performance. As shown in Figure 3, BJ relies on two queues for managing CUs. CUs without any affinity are assigned to the global queue from where they can be pulled from multiple Pilot-Agents. If there is affinity to a certain Pilot because the input data resides in this Pilot-Data, the CU can be placed in a Pilot specific queue. For each CU the following steps are executed:

1. The Pilot-Manager attempts to find a Pilot that best fulfills the requirements of the CU with respect to (i) the requested affinity and (ii) the location of the input data.
2. If a Pilot with the same affinity exists and Pilot has an empty slot, the CU is placed in this pilots queue.
3. If delayed scheduling is active, wait for n sec and recheck whether Pilot has a free slot.
4. If no Pilot is found, the CU is placed in global queue and pulled by first Pilot which has an available slot.

The Pilot-Agent that pulled the CU from a queue is responsible for ensuring that the input DU is staged to the correct location, i. e. before the CU is run, the DU is made available in the working directory of the CU either via remote transfer or a logical link.

In summary, Pilots provide a well-defined abstraction to resources supporting effective application-level/PJ framework level resource management without the necessity to deal with low-level, infrastructure-specific details. Pilot-Abstractions and affinities enable applications as well as the PJ framework runtime system to trade-off different placement options based on the information provided by the affinity assignment, such as resource localities, and dynamic information, such as resource and bandwidth availabilities.

6. EXPERIMENTS

In this section we will establish the importance of the Pilot-Data abstraction by examining two different aspects, viz., (i) the ability to provide uniformity of access and usage modes for different infrastructures (e. g. XSEDE and OSG), (ii) performance advantage arising from the ability to select "optimal" usage modes. Having investigated different strategies of managing compute-data placements in section 3, we explore various compute/data placement strategies using a part of a next-generation sequencing scenario – the read alignment process using BWA – in this section.

BWA Ensemble and Pilot-Abstractions

For this scenario, we utilize the Pilot-API to manage the input/output data and the compute of the BWA genome sequencing applications. The application requires two kinds of input data: (i) the reference genome and index files (∼8 GB) and (ii) the read file(s) (∼2 GB) obtained from the sequencing machines. The alignment process can be parallelized by partitioning the read files and processing them using concurrent BWA instances.

Figure 9 shows the scaling behavior of BWA processing read files with a total volume of 2 GB partitioned across different numbers of Compute-Units. For this experiment we utilize the XSEDE machine Lonestar; for each CU two cores are allocated in the Pilot. The Pilot-Data for the input data is placed co-located with the Pilot-Compute on the Lustre filesystem of Lonestar. The time to completion for this scenario reduces as the number of CUs increases. During the experiments, the Pilot queuing time $T_{Q_{Pilot}}$, i e. the time

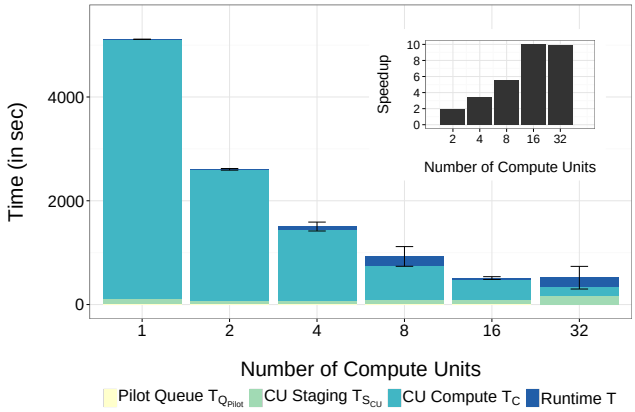

Figure 9: BWA using Pilot-Abstractions on XSEDE/Lonestar: Using concurrent BWA instances, the runtime improves with the increasing number of CUs. The maximum speedup is 10 (see inset); the speedup saturates with >16 CUs as the overhead induced by BigJob and file management increases.

the Pilot waited in the local resource manager's queue, was nearly constant (regardless of the amount of requested resources).

The CU queue time $T_{Q_{CU}}$ describes the coordination overhead for scheduling, queuing and spawning the application process; the CU staging time $T_{S_{CU}}$ is the average time necessary to move the input files to the working directory of the CU. If both data and compute are co-located (as in this scenario), a logical link to the necessary files are created. The more CUs, the more files need to managed; thus, a slight increase in the time is observable as the number of CUs increases. The largest component is the necessary compute time T_C for the BWA application. As shown, the time decreases with the amount of data that needs to be processed by each CU.

In summary, the BWA scenario can be effectively parallelized by distributing the load to multiple compute units. The maximum achievable speed is 10 (see inset of Figure 9); however the efficiency drops with >16 CUs. With the increasing number of concurrent CUs, the ratio between overhead caused by the management of the additional CUs and the actual runtime of the BWA CU becomes unfavorable leading to a reduced speedup with 32 concurrent CUs.

Off-loading of some CUs to distributed/remote resources is a viable strategy to minimize this bottleneck. The main barrier for utilizing distributed resources is the necessity to move the data to this resource. We evaluate when such strategic distribution may be useful and different strategies for utilizing distributed resources in the next section.

Advanced Compute/Data Placement

The aim of this section is to show how system-level and application-level data management based on Pilot-Data can be combined to efficiently manage data and compute in distributed, heterogeneous environments. Infrastructures significantly differ in the way they manage data and compute; for example, on XSEDE [45] it is generally possible to place data on the distributed filesystem mounted to all compute nodes. On OSG [7] this is not simply possible since users generally cannot access compute nodes without Condor;

	Scenario	Compute	Data
1.	OSG/SSH (D2C)	8 PC a 1 core	PD (SSH/Engage)
2.	OSG/iRODS (C2D)	8 PC a 1 core	PD (iRODS)
3.	Lonestar OSG/SSH (hybrid)	1 PC 12 cores 4 PC a 1 core	PD (SSH/Lonestar)
4.	Lonestar (D2C)	1 PC 24 cores	PD (SSH/Lonestar)

Table 1: Pilot-Compute/Pilot-Data Configurations: We run a total of 8 CUs of BWA each processing 8.3 GB data (reference genome & read file). We investigate different Pilot-Compute and Pilot-Data placements on OSG and XSEDE.

however, the iRODS service on OSG enables the application to push data to the different OSG resources. These different kinds of semantics increases the complexity of applications. The Pilot-API and the affinity framework provide applications a tool for mapping different system semantics, e. g. system-level replication vs. no replication support, to a unified, logical resource topology, which allows applications to reason about trade-offs and pursue different compute/data placements strategies as laid out in section 3, e. g. bringing compute to data (C2D) versus data to compute (D2C).

For the purpose of this experiment we utilize BWA to process 2 GB of sequence read files using chunks of 256 MB and 8 CUs. Table 1 summarizes the Pilot-Compute/Pilot-Data configurations used. In scenario 1, we spawn 8 Pilots on OSG (in HTC environments such as OSG, a Pilot can only marshal a single core). The input data is placed in a Pilot-Data on the Engage submission node, i. e. for each CU a remote data transfer between the OSG worker node and the Engage submission node is necessary. Scenario 2 utilizes the same Pilot-Compute setup, but optimizes data placements by using iRODS, which is used to make the input dataset directly accessible at the compute node. Scenario 3 consists of resources located on two different infrastructures: OSG and Lonestar/XSEDE. The input data set resides on a PD on Lonestar; further, we submit 1 PC on Lonestar marshaling 1 node with 12 cores and 4 PCs on OSG. Finally, in scenario 4, both the PC and PD are located on Lonestar.

In all scenarios, the input data set consisting of the index files and the read file is grouped in a DU. In scenario 1, 3 and 4 the DU is placed in a non-replicated, SSH-based Pilot-Data (case A in our model, see section 3), while in scenario 2 we utilize the system-level replication support of iRODS (case B in our model). The OSG Pilot-Computes are submitted using the SAGA-Python Condor adaptor [41] and GlideinWMS [46], a higher-level workload management system built on top of the Pilot capabilities of Condor-G/Glidein [19]. We restrict OSG resources to a set of 9 machines, which are supported by the OSG iRODS installation. The resources are distributed across the eastern and central US including e. g. resources at TACC, Purdue and Cornell. We explore different placement strategies: in scenario 1, the PD is located on a fixed resource and the Pilot-Compute are with locality constraint submitted to OSG (D2C). In scenarios 2 and 4 in contrast, the Pilot-Compute resources are constrained to the location of the data (C2D). Scenario 3 represents a hybrid scenario: we allocate a Pilot-Compute close to the data on Lonestar and additional Pilot-Computes on OSG; the OSG Pilot-Computes are geographically distributed to the Pilot-Data. CUs are bound late to the first Pilot-Compute with a free slot.

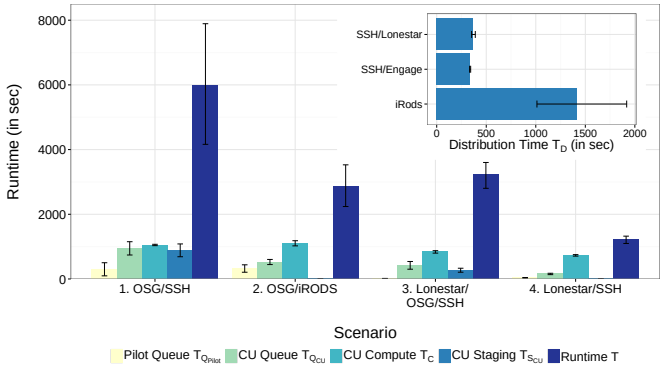

Figure 10: Pilot-based Genome Sequencing on XSEDE and OSG: Runtimes for running BWA on 2 GB of sequence read files using 8 CUs and different infrastructure configurations.

Figure 10 shows the runtime of different scenario consisting of T_Q, T_C and T_D. The insert describes T_D, i. e. the time for uploading, inserting and in case of iRODS replicating 8.3 GB of input data. In general, the Pilot queuing times, i. e. the time until a Pilot becomes active, are higher for OSG than on XSEDE. The queueing time mainly depend on two factors: the current utilization of the resource and the overhead induced by the queuing system. Thus, the higher on average queuing time in the OSG scenarios can be explained by mainly the GlideinWMS system, which must spawn the requested Condor Glide-Ins as well as by the fact that for each core a separate Pilot needs the spawned. That means that for the same scenario on OSG 8 Pilots are started in contrast to 1 on XSEDE.

Another limiting factor in scenario 1 is the necessity to pull in the data remotely. Since OSG resources are remote to the data, the data transfer becomes a bottleneck: for each CU 8.3 GB of data need to be transferred. In scenario 2 we utilize the data/compute co-location capabilities of the OSG Condor and iRODS installation. The runtime T is significantly improved mainly due to the elimination of data transfers. However, the upfront costs for creating the PD and replicating the data across OSG are very high – $T_{D_{iRods}}$ is ~1,418 sec, $T_{D_{SSH}}$ is only ~338 sec for Engage, but does not have a replication component (see inset of Figure 10). Thus, $T_{S_{CU}}$ is significantly higher for SSH than for iRODS. However, even after including T_D, the performance of iRODS is 30 % better than for the SSH scenario.

Scenario 3 investigates the ability to spawn both PCs and Pilot Data Service across multiple infrastructure. Since the input data resides in a PD on Lonestar, the staging time for CUs on Lonestar is significantly reduced. Since the Pilot queuing time on Lonestar was smaller than on OSG, the majority of the CUs have been executed on Lonestar; on average 4.5 out of the 8 CUs are run on Lonestar. Finally, scenario 4 shows that if sufficient compute resources are available close to the data it is beneficial to execute CUs close to the data.

In summary, Pilot-Abstractions enable the interoperable use of different infrastructures as well as the use of different strategies for allocating distributed data and compute resources. The Pilot-Abstractions enable applications to map system-level capabilities, e. g. OSG specifics with respects to

compute handling and the iRODS based replication system, to a unified, logical resource topology, which enables the application to reason about trade-off, such as T_Q vs. T_D for a given amount of compute and data, in order to achieve the best possible performance. While compute-bound applications usually allow the usage of simple heuristics for predicting T_Q on different resources, data-intensive applications require further considerations. Moving data has a significant impact on the overall performance. In particular in the above experiment, where we observed very low Pilot queuing times, bringing compute to data shows a better performance than the other way around.

7. DISCUSSION AND FUTURE WORK

The P* Model suggests new and enhanced usage modes for Pilot-Abstractions. For many of these new modes, as well as existing Pilot-Job usage scenarios, effective data co-placement is, or soon becomes, the barrier. Thus, at the most basic level, Pilot-Data is an associate/adjunct to Pilot-Jobs and enhances the utility and usability of Pilot-Jobs. The advantage of Pilot-Data, however, extends well beyond that: through a series of experiments that cover a range of often realized distributed configurations and scenarios, e. g. bringing compute to data and vice versa, we have seen how Pilot-Data provides a powerful abstraction for distributed data. Pilot-Data combined with Pilot-Jobs provide a unified Pilot-Abstraction, which collectively taken, enables effective management and co-placement of computational tasks and associated data in heterogeneous, distributed environments.

Our focus has been to establish the impact of Pilot-Abstractions on multiple production DCI. These DCI expose different interfaces, middleware and tools; Pilot-Data hides the semantic differences of these backends, and provides a unified abstraction to this vast but often inconsistent data-cyberinfrastructure, and thereby reinforcing the interoperable nature of our Pilot-Abstraction implementations.

Our discussion of affinities suggested they are a good abstraction for capturing relationships between computational tasks and associated data as well as helping map these dependencies to Pilots. Affinities can also be used to map Pilots and resources; in general a multi-level affinity model can be used to enhance multi-level scheduling capabilities, a feature often associated with Pilot-Jobs. Not surprisingly, the Pilot-Abstraction supports the concept of affinity; in fact, an arbitrarily sophisticated model of affinity can be employed in conjunction with the Pilot-Abstraction. By mapping Pilots to resource affinities, applications can reason about performance trade-offs associated with different CU/DU placements. In general, utilizing more than non-trivial resource and application characterization allows for effective scheduling. We will explore some of these themes in future work.

Pilots represent an excellent basis for building higher-level capabilities (e. g. a workload management system or a workflow engine) on distributed infrastructures. Using Pilot-Abstractions, higher-level frameworks can request resources (Pilots), on which they then can run tasks (Compute-Units). In fact, we believe Pilots are powerful enough abstractions that they can serve as the run-time capabilities at the interface of applications and resources. Along with a general purpose model for affinity, examining how Pilot-Abstractions can serve to build higher-level capabilities and middleware for distributed systems is an avenue that we will explore in the near future.

Acknowledgements

This work is funded by an NSF Cyber-enabled Discovery and Innovation Award (CHE-1125332), NSF-ExTENCI (OCI-1007115), NSF EarthCube (SCIHM, OCI-1235085) and Department of Energy Award (ASCR, DE-FG02-12ER26115). This work has also been made possible thanks to computer resources provided by TeraGrid TRAC award TG-MCB090174. EGI experiments were performed on resources provided by SURFsara's Dutch e-Science Grid. their support and input, in particular Andre Merzky, Matteo Turilli and Melissa Romanus. Pradeep Mantha (Berkeley) contributed to early associated work on Pilot-Data based applications. We thank Tanya Levshina for help with iRODS/OSG. We thank J Kim (LSU) for assistance with BWA. SJ acknowledges useful related discussions with Jon Weissman (Minnesota) and Dan Katz (Chicago). SJ acknowledges UK EPSRC via for supporting the e-Science Research themes "Distributed Programming Abstractions" & 3DPAS and earlier support of the SAGA project.

8. REFERENCES

[1] T. Hey, S. Tansley, and K. Tolle, Eds., *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Redmond, Washington: Microsoft Research, 2009. [Online]. Available: http://research.microsoft.com/en-us/collaboration/fourthparadigm/

[2] NSF, "Cyberinfrastructure for 21st century science and engineering: Advanced computing infrastructure vision and strategic plan," http://www.nsf.gov/pubs/2012/nsf12051/nsf12051.pdf, 2012.

[3] S. Jha, N. C. Hong, S. Dobson, D. S. Katz, A. Luckow, O. Rana, Y. Simmhan, and J. Weissman, "Introducing Distributed Dynamic Data-intensive (D3) Science: Understanding Applications and Infrastructure," 2011, http://www.cct.lsu.edu/~sjha/select_publications/3dpas.pdf.

[4] A. Luckow, M. Santcroos, A. Merzky, O. Weidner, P. Mantha, and S. Jha, "P*: A Model of Pilot-Abstractions," in *8th IEEE International Conference on e-Science 2012*, 2012.

[5] A. Luckow, L. Lacinski, and S. Jha, "SAGA BigJob: An Extensible and Interoperable Pilot-Job Abstraction for Distributed Applications and Systems," in *The 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2010, pp. 135–144. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/CCGRID.2010.91

[6] "European grid infrastructure," http://www.egi.eu, 2013.

[7] R. P. *et al*, "The open science grid," *Journal of Physics: Conference Series*, vol. 78, no. 1, p. 012057, 2007.

[8] T. Goodale, S. Jha, H. Kaiser, T. Kielmann, P. Kleijer, A. Merzky, J. Shalf, and C. Smith, "A Simple API for Grid Applications (SAGA)," Open Grid Forum, OGF Recommendation Document, GFD.90, 2007, http://ogf.org/documents/GFD.90.pdf.

[9] Oracle, "Lustre File System: High-Performance Storage Architecture and Scalable Cluster File System (White Paper)," 2007.

[10] "Amazon S3 Web Service," http://s3.amazonaws.com.

[11] W. Allcock, "GridFTP: Protocol Extensions to FTP for the Grid," Open Grid Forum, OGF Recommendation Document, GFD.20, 2003, http://ogf.org/documents/GFD.20.pdf.

[12] A. Sim et al, "GFD.154: The Storage Resource Manager Interface Specification V2.2," Tech. Rep., 2008, global Grid Forum.

[13] A. Rajasekar, R. Moore, C.-y. Hou, C. A. Lee, R. Marciano, A. de Torcy, M. Wan, W. Schroeder, S.-Y. Chen, L. Gilbert, P. Tooby, and B. Zhu, *iRODS Primer: integrated Rule-Oriented Data System.* Morgan and Claypool Publishers, 2010.

[14] A. L. Chervenak, N. Palavalli, S. Bharathi, C. Kesselman, and R. Schwartzkopf, "Performance and scalability of a replica location service," in *Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing*, ser. HPDC '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 182–191. [Online]. Available: http://dx.doi.org/10.1109/HPDC.2004.27

[15] J.-P. Baud, J. Casey, S. Lemaitre, and C. Nicholson, "Performance analysis of a file catalog for the lhc computing grid," in *High Performance Distributed Computing, 2005. HPDC-14. Proceedings. 14th IEEE International Symposium on*, july 2005, pp. 91 – 99.

[16] I. Foster, "Globus online: Accelerating and democratizing science through cloud-based services," *IEEE Internet Computing*, vol. 15, pp. 70–73, 2011.

[17] "The grid information system," https://tomtools.cern.ch/confluence/display/IS/Home, 2013.

[18] GFFS – Global Federated File System, http://genesis2.virginia.edu/wiki/Main/GFFS.

[19] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids," *Cluster Computing*, vol. 5, no. 3, pp. 237–246, July 2002.

[20] J. Moscicki, "Diane - distributed analysis environment for grid-enabled simulation and analysis of physics data," in *Nuclear Science Symposium Conference Record, 2003 IEEE*, vol. 3, 2003, pp. 1617 – 1620.

[21] P.-H. Chiu and M. Potekhin, "Pilot factory – a condor-based system for scalable pilot job generation in the panda wms framework," *Journal of Physics: Conference Series*, vol. 219, no. 6, p. 062041, 2010.

[22] "Topos - a token pool server for pilot jobs," https://grid.sara.nl/wiki/index.php/Using_the_Grid/ToPoS, 2011.

[23] R. Buyya, D. Abramson, and J. Giddy, "Nimrod/g: An architecture for a resource management and scheduling system in a global computational grid," *International Conference on High-Performance Computing in the Asia-Pacific Region*, vol. 1, pp. 283–289, 2000.

[24] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, and M. Wilde, "Falkon: A Fast and Light-Weight TasK ExecutiON Framework," in *SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing.* New York, NY, USA: ACM, 2007, pp. 1–12.

[25] E. Walker, J. Gardner, V. Litvin, and E. Turner, "Creating personal adaptive clusters for managing scientific jobs in a distributed computing environment," in *Challenges of Large Applications in Distributed Environments, 2006 IEEE*, 0-0 2006, pp. 95–103.

[26] A. Tsaregorodtsev, N. Brook, A. Casajus Ramo, P. Charpentier, J. Closier *et al.*, "DIRAC3: The new generation of the LHCb grid software," *J.Phys.Conf.Ser.*, vol. 219, p. 062029, 2010.

[27] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, and I. Foster, "Swift: A language for distributed parallel scripting," *Parallel Computing*, vol. 37, no. 9, pp. 633–652, 2011.

[28] T. Kosar and M. Livny, "Stork: making data placement a first class citizen in the grid," in *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on*, 2004, pp. 342 – 349.

[29] A. Romosan, D. Rotem, A. Shoshani, and D. Wright, "Co-scheduling of computation and data on computer clusters," in *Proceedings of 17th International Conference on Scientific and Statistical Databases Management (SSDBM)*, 2005.

[30] "Apache Hadoop," http://hadoop.apache.org/, 2013.

[31] "Apache Hadoop NextGen MapReduce (YARN)," http://hadoop.apache.org/docs/r0.23.0/hadoop-yarn/hadoop-yarn-site/YARN.html, 2013.

[32] A. Aamnitchi, S. Doraimani, and G. Garzoglio, "Filecules in high-energy physics: Characteristics and impact on resource management," in *High Performance Distributed Computing, 2006 15th IEEE International Symposium on*, 0-0 2006, pp. 69 –80.

[33] A. Amer, D. Long, and R. Burns, "Group-based management of distributed file caches," in *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, 2002, pp. 525 – 534.

[34] G. Ganger and M. F. Kaashoek, "Embedded inodes and explicit grouping: Exploiting disk bandwidth for small files," in *In Proceedings of the 1997 USENIX Technical Conference*, 1997, pp. 1–17.

[35] G. Fedak, H. He, and F. Cappello, "Bitdew: a programmable environment for large-scale data management and distribution," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, ser. SC '08. Piscataway, NJ, USA: IEEE Press, 2008, pp. 45:1–45:12. [Online]. Available: http://dl.acm.org/citation.cfm?id=1413370.1413416

[36] M. D. Beynon, T. Kurc, U. Catalyurek, C. Chang, A. Sussman, and J. Saltz, "Distributed processing of very large datasets with datacutter," *Parallel Comput.*, vol. 27, no. 11, pp. 1457–1478, Oct. 2001. [Online]. Available: http://dx.doi.org/10.1016/S0167-8191(01)00099-0

[37] K. Ranganathan and I. Foster, "Decoupling computation and data scheduling in distributed data-intensive applications," in *High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings. 11th IEEE International Symposium on*, 2002, pp. 352 – 358.

[38] W. H. Bell, D. G. Cameron, A. P. Millar, L. Capozza, K. Stockinger, and F. Zini, "Optorsim: A grid simulator for studying dynamic data replication strategies," *International Journal of High Performance Computing Applications*, vol. 17, no. 4, pp. 403–416,

2003. [Online]. Available: http://hpc.sagepub.com/content/17/4/403.abstract

[39] R. Mukherjee, A. Thota, H. Fujioka, T. C. Bishop, and S. Jha, "Running many molecular dynamics simulations on many supercomputers," in *Proceedings of the 1st Conference of the Extreme Science and Engineering Discovery Environment: Bridging from the eXtreme to the Campus and Beyond*, ser. XSEDE '12. New York, NY, USA: ACM, 2012, pp. 2:1–2:9.

[40] Pilot API, http://saga-project.github.com/BigJob/apidoc/.

[41] SAGA Python, https://github.com/saga-project/bliss/wiki.

[42] P. K. Mantha, A. Luckow, and S. Jha, "Pilot-MapReduce: An Extensible and Flexible MapReduce Implementation for Distributed Data," in *Proceedings of third international workshop on MapReduce and its Applications*, ser. MapReduce '12. New York, NY, USA: ACM, 2012, pp. 17–24.

[43] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, Nov. 1994. [Online]. Available: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0201633612

[44] C. Miceli, M. Miceli, B. Rodriguez-Milla, and S. Jha, "Understanding Performance of Distributed Data-Intensive Applications," *Royal Society of London Philosophical Transactions Series A*, vol. 368, pp. 4089–4102, Aug. 2010.

[45] XSEDE: Extreme Science and Engineering Discovery Environment, https://www.xsede.org/.

[46] I. Sfiligoi, D. Bradley, B. Holzman, P. Mhashilkar, S. Padhi, and F. Wurthwein, "The pilot way to grid resources using glideinwms," in *Computer Science and Information Engineering*, 2009, pp. 428 –432.